Flutter Codes

Question 1:Use SafeArea Widget in flutter App development

```dart
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: SafeAreaDemo(),
    );
  }
}

class SafeAreaDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SafeArea( // ⬅️ Wrap your content inside SafeArea
        child: Center(
          child: Text(
            'This text is inside SafeArea!',
            style: TextStyle(fontSize: 20),
          ),
        ),
      ),
    );
  }
}
```

The SafeArea widget ensures that the content of the app is not hidden behind system status bars, notches, or navigation areas. It's especially useful for full-screen UIs on modern devices with display cutouts.

Question 2:Use AppBar Widget in flutter App development

```dart
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: AppBarDemo(),
    );
  }
}

class AppBarDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar( // ⬅️ Using AppBar inside Scaffold
        title: Text('My AppBar Title'),
        backgroundColor: Colors.deepPurple,
        actions: [
          IconButton(
            icon: Icon(Icons.settings),
            onPressed: () {
              // Action when settings icon is pressed
            },
          )
        ],
      ),
      body: Center(
        child: Text('Hello from body!'),
      ),
    );
  }
}
```

The AppBar widget creates a material design app bar at the top of the screen. It usually includes a title, and optionally icons like a back button or actions like settings or search.

Question 3:Use Row Widget in flutter App development

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: RowDemo(),
    );
  }
}

class RowDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Row Widget')),
      body: Center(
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: [
            Icon(Icons.home, size: 40, color: Colors.blue),
            Icon(Icons.favorite, size: 40, color: Colors.red),
            Icon(Icons.person, size: 40, color: Colors.green),
          ],
        ),
      ),
    );
  }
}
```

The Row widget is used to display its children in a horizontal direction. We can align the children using properties like mainAxisAlignment and crossAxisAlignment.

Question 4:Use Column Widget in flutter App development

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: ColumnDemo(),
    );
  }
}

class ColumnDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Column Widget')),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Icon(Icons.phone, size: 40, color: Colors.blue),
            SizedBox(height: 10),
            Text('Call', style: TextStyle(fontSize: 18)),
          ],
        ),
      ),
    );
  }
}
```

The Column widget arranges its children vertically. We can control alignment with properties like mainAxisAlignment and crossAxisAlignment. It's commonly used to stack elements top-to-bottom.

Question 5:-Use Container Widget with child property in flutter App development

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: ContainerDemo(),
    );
  }
}
```

```
class ContainerDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Container Widget')),
      body: Center(
        child: Container(
          padding: EdgeInsets.all(20),
          color: Colors.amber,
          child: Text(
            'Inside Container!',
            style: TextStyle(fontSize: 20),
          ),
        ),
      ),
    );
  }
}
```

The Container widget is a box model widget that can have padding, margin, color, size, and more. The child property is used to place another widget inside it, like Text, Row, etc.

6.Use Container Widget with color property in flutter App development

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: ColorContainerDemo(),
    );
  }
}

class ColorContainerDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Container with Color')),
      body: Center(
        child: Container(
          width: 150,
          height: 150,
          color: Colors.teal, // ⬅️ Color property
        ),
      ),
    );
  }
}
```

The color property in a Container is used to set its background color. It works only if the Container has no decoration or if it's used within BoxDecoration.

7. Use Container Widget with height and width property in flutter App development

```dart
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: SizeContainerDemo(),
    );
  }
}

class SizeContainerDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Container Size')),
      body: Center(
        child: Container(
          height: 100,
          width: 200,
          color: Colors.orange,
          child: Center(
            child: Text('Sized Container'),
          ),
        ),
      ),
    );
  }
}
```

The height and width properties of a Container define its size. If these are not set, the container will try to size itself based on its child or parent constraints.

8.Use Container Widget with margin property in flutter App development

```dart
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: MarginContainerDemo(),
    );
  }
}

class MarginContainerDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Container with Margin')),
      body: Center(
        child: Container(
          margin: EdgeInsets.all(20), // ⬅️ Adds space around the
container
          color: Colors.lightBlue,
          height: 100,
          width: 200,
          child: Center(child: Text('With Margin')),
        ),
      ),
    );
  }
}
```

The margin property adds space outside the container’s boundary. It's like adding breathing room around the widget. You can use EdgeInsets.all, only, symmetric, etc.

9.Use Container Widget with padding property in flutter App development

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: PaddingContainerDemo(),
    );
  }
}

class PaddingContainerDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Container with Padding')),
      body: Center(
        child: Container(
          padding: EdgeInsets.all(20), // ⬅️ Adds space inside the
container
          color: Colors.purpleAccent,
          child: Text(
            'With Padding',
            style: TextStyle(fontSize: 18, color: Colors.white),
          ),
        ),
      ),
    );
  }
}
```

The padding property adds space inside the container, between its child and its edges. It’s useful for giving content breathing space within the container.

10.Use Container Widget with alignment property in flutter App development

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: AlignContainerDemo(),
    );
  }
}

class AlignContainerDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Container with Alignment')),
      body: Center(
        child: Container(
          height: 150,
          width: 150,
          color: Colors.teal,
          alignment: Alignment.bottomRight, // ⬅️ Aligns child inside the container
          child: Text(
            'Bottom Right',
            style: TextStyle(color: Colors.white),
          ),
        ),
      ),
    );
  }
}
```

The alignment property controls how the child is positioned inside the container. For example, Alignment.center, topLeft, bottomRight, etc., align the child accordingly.

11. Use Container Widget with decoration property in flutter App development

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());
```

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: DecorationContainerDemo(),
    );
  }
}

class DecorationContainerDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Container with Decoration')),
      body: Center(
        child: Container(
          height: 150,
          width: 150,
          decoration: BoxDecoration(
            color: Colors.blue,
            borderRadius: BorderRadius.circular(12),
            boxShadow: [
              BoxShadow(
                color: Colors.black26,
                blurRadius: 8,
                offset: Offset(4, 4),
              ),
            ],
          ),
          child: Center(
            child: Text(
              'Decorated',
              style: TextStyle(color: Colors.white, fontWeight:
FontWeight.bold),
            ),
          ),
        ),
      ),
    );
  }
}
```

The decoration property is used to style the Container. It allows setting background color, rounded corners, gradients, shadows, borders, and more using BoxDecoration.

12.Use Container Widget with transform property in flutter App development

```
import 'package:flutter/material.dart';
import 'dart:math'; // Required for pi

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: TransformContainerDemo(),
    );
  }
}

class TransformContainerDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Container with Transform')),
      body: Center(
        child: Container(
          width: 150,
          height: 150,
          color: Colors.green,
          transform: Matrix4.rotationZ(pi / 12), // Rotates container
slightly
          alignment: Alignment.center,
          child: Text(
            'Rotated Box',
            style: TextStyle(color: Colors.white, fontWeight:
FontWeight.bold),
          ),
        ),
      ),
    );
  }
}
```

The transform property allows you to apply 2D or 3D transformations to the container, like rotation, scaling, or translation. Here, Matrix4.rotationZ rotates the container around the Z-axis.

13.Use FlatButton Widget in flutter App development
FlatButton is deprecated in newer versions of Flutter, replaced by TextButton

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: TextButtonDemo(),
    );
  }
}

class TextButtonDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('TextButton Example')),
      body: Center(
        child: TextButton(
          style: TextButton.styleFrom(
            primary: Colors.white, // Text color
            backgroundColor: Colors.blue, // Background color
          ),
          onPressed: () {
            print('TextButton Pressed!');
          },
          child: Text('Click Me'),
        ),
      ),
    );
  }
}
```

The TextButton widget is used to create a text-based button. It is lightweight, customizable, and doesn't have an elevation by default. You can style it with properties like primary for text color and backgroundColor for button color.

14.Use RaisedButton Widget in flutter App development
RaisedButton is deprecated in newer Flutter versions, replaced by ElevatedButton

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: RaisedButtonDemo(),
    );
  }
}

class RaisedButtonDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('RaisedButton Example')),
      body: Center(
        child: RaisedButton(
          color: Colors.green, // Button background color
          textColor: Colors.white, // Text color
          onPressed: () {
            print('RaisedButton Pressed!');
          },
          child: Text('Click Me'),
        ),
      ),
    );
  }
}
```

The RaisedButton widget creates a button with a material "elevated" effect. However, it has been deprecated in favor of ElevatedButton starting from Flutter 2.0, which serves the same purpose but with better support and flexibility.

15. Use Floating Action Button Widget in flutter App development

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: FloatingActionButtonDemo(),
    );
  }
}

class FloatingActionButtonDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('FloatingActionButton Example')),
      body: Center(
        child: Text('Press the Floating Action Button!'),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          print('Floating Action Button Pressed!');
        },
        child: Icon(Icons.add), // Icon displayed on the FAB
        backgroundColor: Colors.blue, // Background color of the FAB
      ),
    );
  }
}
```

The FloatingActionButton (FAB) is a button that floats above the content, typically used for primary actions in an app. You can customize its size, shape, color, and the icon inside it. It's most often positioned in the bottom right corner by default.

16. Use Dropdown Button Widget in flutter App development

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: DropdownButtonDemo(),
    );
  }
}

class DropdownButtonDemo extends StatefulWidget {
  @override
  _DropdownButtonDemoState createState() => _DropdownButtonDemoState();
}

class _DropdownButtonDemoState extends State<DropdownButtonDemo> {
  String dropdownValue = 'One'; // Initial value of the dropdown

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('DropdownButton Example')),
      body: Center(
        child: DropdownButton<String>(
          value: dropdownValue, // Current selected value
          onChanged: (String? newValue) {
            setState(() {
              dropdownValue = newValue!; // Update selected value
            });
          },
          items: <String>['One', 'Two', 'Three', 'Four']
              .map<DropdownMenuItem<String>>((String value) {
            return DropdownMenuItem<String>(
              value: value,
              child: Text(value),
            );
          }).toList(),
        ),
      ),
    );
  }
}
```

The DropdownButton widget allows you to create a dropdown list of items. You can define the list of options through the items property, and the selected value is managed using a variable. The onChanged callback handles when the user selects a new item.

17. Use Icon Button Widget in flutter App development

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: IconButtonDemo(),
    );
  }
}

class IconButtonDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('IconButton Example')),
      body: Center(
        child: IconButton(
          icon: Icon(Icons.favorite), // The icon you want to show
          color: Colors.red, // Icon color
          onPressed: () {
            print('IconButton Pressed!');
          },
        ),
      ),
    );
  }
}
```

The IconButton widget creates a clickable button that displays an icon. You can customize the icon's appearance, color, and size. The onPressed callback is triggered when the button is clicked.

18. Use RichText Widget in flutter App development

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: RichTextDemo(),
    );
  }
}

class RichTextDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('RichText Example')),
      body: Center(
        child: RichText(
          text: TextSpan(
            children: <TextSpan>[
              TextSpan(
                text: 'Hello, ',
                style: TextStyle(color: Colors.black, fontSize: 20),
              ),
              TextSpan(
                text: 'Flutter!',
                style: TextStyle(color: Colors.blue, fontSize: 30,
fontWeight: FontWeight.bold),
              ),
              TextSpan(
                text: ' Welcome to RichText widget.',
                style: TextStyle(color: Colors.green, fontSize: 18),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

The RichText widget is used to display text with multiple styles. You define the text as a TextSpan and apply different styles to parts of the text using the style property. This allows for different font sizes, colors, weights, etc., within the same block of text.

19.Use Container Widget in flutter App development.
var kelay solve

20.Use GridView Widget in flutter App development.

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: GridViewDemo(),
    );
  }
}

class GridViewDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('GridView Example')),
      body: GridView.count(
        crossAxisCount: 3, // Number of columns
        padding: EdgeInsets.all(10),
        children: <Widget>[
          Container(
            color: Colors.red,
            child: Center(child: Text('Item 1')),
          ),
          Container(
            color: Colors.green,
            child: Center(child: Text('Item 2')),
          ),
          Container(
            color: Colors.blue,
            child: Center(child: Text('Item 3')),
          ),
          Container(
            color: Colors.orange,
            child: Center(child: Text('Item 4')),
          ),
          Container(
            color: Colors.purple,
            child: Center(child: Text('Item 5')),
          ),
          Container(
            color: Colors.yellow,
            child: Center(child: Text('Item 6')),
          ),
        ],
      ),
    );
  }
}
```

The GridView widget is used to create a scrollable grid of items. In this example, we used GridView.count to specify the number of columns (crossAxisCount) in the grid. Each item in the grid is wrapped inside a Container with different background colors. You can also use other types of GridView constructors like GridView.builder or GridView.extent based on specific needs.

21.Use ListView Widget in flutter App development.

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: ListViewDemo(),
    );
  }
}

class ListViewDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('ListView Example')),
      body: ListView(
        children: <Widget>[
          ListTile(
            leading: Icon(Icons.home),
            title: Text('Home'),
            onTap: () {
              print('Home tapped!');
            },
          ),
          ListTile(
            leading: Icon(Icons.person),
            title: Text('Profile'),
            onTap: () {
              print('Profile tapped!');
            },
          ),
          ListTile(
            leading: Icon(Icons.settings),
            title: Text('Settings'),
            onTap: () {
              print('Settings tapped!');
            },
          ),
          ListTile(
            leading: Icon(Icons.info),
            title: Text('About'),
            onTap: () {
              print('About tapped!');
            },
          ),
        ],
      ),
    );
  }
}
```

The ListView widget is used to display a scrolling list of widgets. Each list item in this example is a ListTile, which is a material design component that typically contains an icon and text. The ListView automatically handles scrolling for a large number of items. This example demonstrates a basic static list, but you can also use ListView.builder for dynamic lists when the items are generated at runtime.

22. Use StackView Widget in flutter App development.

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: StackDemo(),
    );
  }
}

class StackDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Stack Widget Example')),
      body: Center(
        child: Stack(
          alignment: Alignment.center, // Align all children to the
center of the stack
          children: <Widget>[
            Container(
              width: 200,
              height: 200,
              color: Colors.blue,
            ),
            Positioned(
              top: 20,
              left: 20,
              child: Container(
                width: 100,
                height: 100,
                color: Colors.red,
              ),
            ),
            Positioned(
              bottom: 20,
              right: 20,
              child: Container(
                width: 50,
                height: 50,
                color: Colors.green,
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

The Stack widget allows you to overlay widgets on top of each other. It is very useful for creating layered UI elements. In this example, there is a blue container at the bottom, a red container positioned at the top left, and a green container positioned at the bottom right using the Positioned widget. The alignment property can be used to align all children within the stack.

23 .Use CardView Widget in flutter App development.

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: CardDemo(),
    );
  }
}

class CardDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Card Widget Example')),
      body: Center(
        child: Card(
          elevation: 5, // Adds shadow to the card
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(15), // Rounded corners
          ),
          child: Padding(
            padding: const EdgeInsets.all(16.0),
            child: Column(
              mainAxisSize: MainAxisSize.min,
              children: <Widget>[
                Text(
                  'Flutter Card View',
                  style: TextStyle(fontSize: 24, fontWeight:
FontWeight.bold),
                ),
                SizedBox(height: 10),
                Text(
                  'This is a simple card view using Flutter\'s Card
widget.',
                  style: TextStyle(fontSize: 16),
                  textAlign: TextAlign.center,
                ),
              ],
            ),
          ),
        ),
      ),
    );
  }
}
```

The Card widget is used to create a material design card with a shadow and rounded corners. In this example, the elevation property gives the card a shadow effect, and the shape property is used to round the corners of the card. Inside the card, we use Padding to add spacing around the text, and Column to arrange multiple widgets vertically. The Card widget is commonly used for displaying grouped content with a card-like appearance.

24.Use ListTile Widget in flutter App development.

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: ListTileDemo(),
    );
  }
}

class ListTileDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('ListTile Widget Example')),
      body: ListView(
        children: <Widget>[
          ListTile(
            leading: Icon(Icons.home),
            title: Text('Home'),
            subtitle: Text('Go to the home page'),
            onTap: () {
              print('Home tapped!');
            },
          ),
          ListTile(
            leading: Icon(Icons.person),
            title: Text('Profile'),
            subtitle: Text('View your profile'),
            onTap: () {
              print('Profile tapped!');
            },
          ),
          ListTile(
            leading: Icon(Icons.settings),
            title: Text('Settings'),
            subtitle: Text('Adjust app settings'),
            onTap: () {
              print('Settings tapped!');
            },
          ),
          ListTile(
            leading: Icon(Icons.help),
            title: Text('Help'),
            subtitle: Text('Get help and support'),
            onTap: () {
              print('Help tapped!');
            },
          ),
        ],
      ),
    );
  }
}
```

The ListTile widget is used to create a single item in a list, typically containing text and an icon. In this example, each ListTile contains a leading icon, a title, and a subtitle. The onTap callback allows for interaction with each list item. This is useful for creating list-based UIs, like settings screens or navigation menus.

25.Use flutter icons in flutter App development.

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: IconDemo(),
    );
  }
}

class IconDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Flutter Icons Example')),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Icon(
              Icons.home, // Flutter's built-in icon for home
              size: 50.0,  // Icon size
              color: Colors.blue,  // Icon color
            ),
            SizedBox(height: 20),
            Icon(
              Icons.favorite, // Flutter's built-in icon for favorite
              size: 50.0,
              color: Colors.red, // Icon color
            ),
            SizedBox(height: 20),
            Icon(
              Icons.search, // Flutter's built-in icon for search
              size: 50.0,
              color: Colors.green, // Icon color
            ),
          ],
        ),
      ),
    );
  }
}
```

In this example, I have used the Icon widget in Flutter to display various built-in icons such as Icons.home, Icons.favorite, and Icons.search. Each icon's size and color are customized using the size and color properties. Flutter comes with a wide variety of built-in icons, which are part of the Icons class. You can choose from over 1,000 material design icons and customize them as needed for your app's design.

26.Use flutter image.asset to add image in flutter App development.

To add an image using Image.asset in Flutter, you need to first add the image to your project's assets folder and then declare it in the pubspec.yaml file. After that, you can display the image using the Image.asset widget.

Steps:
Add image to the assets folder:

Create an assets folder in the root of your Flutter project.

Place the image file (e.g., my_image.png) inside the assets folder.

Declare the image in pubspec.yaml:

Open your pubspec.yaml file and make sure to declare the assets path like this:

yaml

Copy
Edit
```
flutter:
  assets:
    - assets/my_image.png
```
Use Image.asset to display the image:

Now you can use Image.asset to load and display the image in your app.

code:-

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: ImageAssetDemo(),
    );
  }
}

class ImageAssetDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Image.asset Example')),
      body: Center(
        child: Image.asset(
          'assets/my_image.png',  // Path to the image
          width: 200,  // Optional width
          height: 200,  // Optional height
        ),
      ),
    );
  }
}
```

In this example, I use Image.asset to load and display an image from the app's assets folder. The image file my_image.png is declared in the pubspec.yaml file under the assets section. I also set the width and height properties to control the image's size. This method is commonly used to display static images bundled with the app, such as logos or background images.

27. Use Flutter Navigation and Routing in flutter App development.

In Flutter, navigation and routing allow you to move between different screens (or routes) within your app. You can use Flutter’s built-in Navigator and Route classes to handle routing.

Steps:
1)Define multiple screens (or routes).

2)Navigate between screens using Navigator.push or Navigator.pushNamed.

3)Use Navigator.pop to return to the previous screen.

code:-
```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      initialRoute: '/', // Default route
      routes: {
        '/': (context) => HomeScreen(),
        '/second': (context) => SecondScreen(),
      },
    );
  }
}

class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Home Screen')),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            // Navigating to the SecondScreen using named route
            Navigator.pushNamed(context, '/second');
          },
          child: Text('Go to Second Screen'),
        ),
      ),
    );
  }
}

class SecondScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Second Screen')),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            // Navigating back to the previous screen
            Navigator.pop(context);
          },
          child: Text('Back to Home Screen'),
        ),
      ),
    );
  }
}
```

This example demonstrates basic navigation and routing in Flutter using Navigator.pushNamed and Navigator.pop. The app starts with the HomeScreen, where the user can tap a button to navigate to the SecondScreen using the named route /second. On the second screen, the user can tap a button to return to the home screen using Navigator.pop. The routes are defined in the MaterialApp widget using the routes property, and the default route ('/') is set to the HomeScreen.

Key Concepts:
Navigator.pushNamed: Used to navigate to a new screen using a named route.

Navigator.pop: Used to pop the current screen off the navigation stack and return to the previous screen.

MaterialApp.routes: A map of route names to route builder functions used to define named routes for navigation.

29. Use flutter to implement Gesture in App development.

In Flutter, you can handle user gestures (like taps, swipes, and long presses) using GestureDetector or other specific gesture widgets like InkWell or RawGestureDetector. The GestureDetector widget is commonly used for detecting basic gestures such as tap, double-tap, long press, and pan.

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: GestureDemo(),
    );
  }
}

class GestureDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('GestureDetector Example')),
      body: Center(
        child: GestureDetector(
          onTap: () {
            print('Tapped!');
          },
          onDoubleTap: () {
            print('Double Tapped!');
          },
          onLongPress: () {
            print('Long Pressed!');
          },
          onPanUpdate: (details) {
            print('Dragging at position: ${details.localPosition}');
          },
          child: Container(
            color: Colors.blue,
            width: 200,
            height: 200,
            child: Center(
              child: Text(
                'Tap Me!',
                style: TextStyle(color: Colors.white, fontSize: 20),
              ),
            ),
          ),
        ),
      ),
    );
  }
}
```

In this example, I’ve used the GestureDetector widget to handle various gestures in Flutter. The GestureDetector allows me to listen for taps, double taps, long presses, and even dragging events. When a gesture is detected, a corresponding action is performed (like printing a message to the console). The widget is wrapped around a Container, and I defined actions for each gesture:

onTap is triggered when the user taps the container.

onDoubleTap occurs when the user double-taps the container.

onLongPress is triggered by a long press.

onPanUpdate gives the position of the drag movement.

The GestureDetector is a powerful tool in Flutter for adding interaction to your widgets.

Key Concepts:
onTap: Triggered when the user taps the widget.

onDoubleTap: Triggered when the user double taps the widget.

onLongPress: Triggered when the user presses and holds the widget for a longer period.

onPanUpdate: Captures the movement of a user's finger as they drag across the screen.